Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Absolvování individuální odborné praxe

Individual professional practice in the company

2020 **Adrián Mindek**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Adrián Mindek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1.Student vykoná individuální praxi ve firmě: MORAVIO, s.r.o.
2.Struktura závěrečné zprávy:
a.Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b.Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c.Zvolený postup řešení zadaných úkolů
d.Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e.Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f.Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeňka Chmelíková, Ph.D.**

Konzultant bakalářské práce: Bc. Daniel Karchňák

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

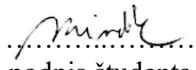
prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave dňa: *14. mája 2020*



podpis študenta

Poďakovanie

Rád by som poďakoval firme Moravio, s.r.o., Bc. Danielovi Karchňákovi a Ing. Šárke Skopalovej za vytvorenie príjemnej atmosféry, odbornú pomoc a konzultácie v priebehu vykonávania tejto odbornej praxe. Zároveň ďakujem Ing. Zdenke Chmelíkovej, PhD. za odborné konzultácie pri tvorbe mojej práce.

Prehlásenie zástupcu spolupracujúcej právnickej alebo fyzickej osoby

„Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava."

Dne: 14. 5. 2020

.....
Ing. Šárka Skopalová

Abstrakt

V tejto bakalárskej práci popisujem priebeh mojej odbornej praxe vo firme Moravio, s.r.o. V práci popisujem stručne odborné zameranie firmy. Ďalej opisujem technológie, ktoré som počas praxe využíval a úlohy, ktoré som riešil počas môjho pôsobenia vo firme. Pracoval som na dvoch projektoch a v oboch som riešil niekoľko úloh. Každá úloha obsahuje popis zadania a postupu pri jej riešení. Počas praxe som sa naučil veľa nových vývojárskych technológií a nadobudol som mnoho nových skúseností.

Kľúčové slová

Individuálna odborná prax; informačný systém; PHP; Laravel; JavaScript; Vue.js;

Abstract

In this bachelor thesis I am describing the process of my professional practice in the company Moravio, s.r.o. In this thesis I am describing briefly the description and expertise focus of the company. Further on I am describing technologies that I was using during the practice and the tasks which I was solving during my work in the company. I was working on two projects and in both of them I was solving several tasks. Each task contains the description of the process of how I was proceeding in making of their solutions. During the practice I learned a lot of new development technologies and I gained many new experiences.

Key words

Individual professional practice; information system; PHP; Laravel; JavaScript; Vue.js;

Zoznam použitých skratiek

Skratka	Význam
MVC	Model-View-Controller
CLI	Command line interface
CRUD	Create-Read-Updare-Delete
SQL	Structured Query Language
OS	Operačný systém
Id	Identifikátor
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
URL	Uniform Resource Locator

Zoznam použitých termínov

Termín	Význam termínu
PHP	Programovací jazyk bežiaci na strane servera
Framework	Programová knižnica, má za úlohu uľahčiť prácu programátorovi
WordPress	Populárny PHP framework
Design na mieru	Dizajn podľa požiadaviek zákazníka
Slack	Aplikácia slúžiaca na komunikáciu medzi členmi tímu
JavaScript	Programovací jazyk
Laravel	PHP framework
Backend	Kód bežiaci na pozadí, na strane servera
Frontend	Kód bežiaci v popredí, na strane klienta
Routing	Smerovanie
Factories	Továrne (na dáta)
Eloquents	Programové rozhranie Laravelu pre prácu s databázou
Vue.js	JavaScript framework
Component	Časť Vue.js kódu utvárajúca jeden menší celok (napr. tlačidlo)
SearchBox	Formulárový vstup od užívateľa slúžiaci na vyhľadávanie
Mikroservica	Menšia funkcionalita (napr. Správa užívateľov)
Docker	Služba umožňujúca virtualizáciu
Docker-image	Stav virtualizácie, niečo ako uloženie dokumentu
Homestead	Prednastavené oficiálne riešenie pre vývoj vo frameworku Laravel
MySQL	Databázový relačný systém
SQL jazyk	Dotazovací jazyk pre SQL databázové servery
phpMyAdmin	Webový klient pre správu databázy
MacOS	Operačný systém pre Apple zariadenia
Sequel Pro	Klient pre pripojenie a správu databázy pre počítače s MacOS
Linux	Typ operačného systému
Grid	Mriežka

Produkčná databáza	Ostrá databáza, používaná v reálnej verzii
Docker container	Časť virtualizácie, v ktorej beží jedna služba
Zaseedovať databázu	Naplniť databázu dátami
Http request	Požiadavka na server vo forme http protokolu
Coodevelopment	Práca na projekte s inou firmou
Jira	Nástroj na správu úloh v tíme
Git	Systém riadenia revízií
Gitlab	Webový Git repozitár
Slack	Chatovací softvér pre vývojárov softvéru
Basecamp	Softvér pre komunikáciu medzi vývojármi
API rozhranie	Rozhranie pre programovanie aplikácií
React	JavaScriptovský framework
Angular	JavaScriptovský framework
Node.js	Softverový systém navrhnutý pre prácu s JavaScriptom aj mimo prehliadač (napr. na Serveri)
App key	Aplikačný kľúč
Debugger	Nástroj na hľadanie chýb pri programovaní
Seeder	Trieda slúžiaca na automatické napĺňanie dát v databáze
VirtualBox	Multiplatformový virtualizačný nástroj
Http request	Požiadavka vo formáte http protokolu posielaná na server
Select	Komponent slúžiaci na výber jedného prvku zo zoznamu
MultiSelect	Komponent slúžiaci na výber viacerých prvkov zo zoznamu
Text filter	Vstup filtrujúci na základe zadaného textu od užívateľa
Datepicker	Komponent slúžiaci na výber dátumu (filter)
Checkbox	Komponent umožňujúci zaškrtnutie/odškrtnutie jednej alebo viacerých možností
Bootstrap	CSS framework na tvorbu užívateľských rozhraní
Widget	Ovládací prvok
Layout	Grafické rozloženie (stránky)
Middleware	Softvérová vrstva medzi programovými komponentmi

Service layer pattern	Návrhový vzor aplikácií
Sparsovať dáta	Spracovať dáta z textového vstupu
JSON	Dátový formát
Base metódy	Základné metódy
Modal	Vyskakovacie okno bez nutnosti refreshu/obnovy stránky
Select2.js	JavaScriptovská knižnica pre prácu so Select komponentami
BootstrapVue	Nadstavba frameworku Vue.js
Chunk	Časť celku (ktorý sa spracováva alebo renderuje)
Event	Udalosť v programe
Interface	Programová konštrukcia
Carbon	Knižnica pre prácu s dátumami na strane servera pomocou PHP
Chart.js	JavaScriptovská knižnica pre prácu s grafmi

Obsah

Úvod

V tejto bakalárskej práci popisujem priebeh mojej odbornej praxe vo firme Moravio, s.r.o. Odbornú prax som si zvolil z dôvodu, že na trhu je stále väčší a väčší záujem o absolventov s praktickými skúsenosťami namiesto tých iba s teoretickými znalosťami. S týmto názorom úplne súhlasím, keďže každá práca môže byť iná a nie je úplne možné, aby sa teoreticky dalo pripraviť na všetko. Prax je preto skvelá príležitosť ako sa učiť od skúsenejších kolegov, čo som mal možnosť aj sám vyskúšať. Za podstatný dôvod voľby odbornej praxe považujem možnosť vyvíjať produkt pre reálnych klientov, ktorý im bude slúžiť nejakú dobu.

Počas stáže som sa zoznámil s nemálo novými technológiami a pracoval som na niekoľkých projektoch ako vývojár. Tieto projekty by sa dali rozdeliť na 3 časti. Do prvej časti by som zaradil informačný systém pre klienta, ktorý sa zaoberá vývozom a obchodovaním so stavebným odpadom. Do tohto projektu som sa počas mojej praxe zapojil najviac. Dostal som na starosť najmä jeden väčší modul, ktorému sa budem venovať v tejto práci neskôr. Do druhej, o niečo menšej časti, by som zaradil prácu na ekonomickom module, do už pred tým vytvoreného informačného systému pre pracovnú agentúru. No a v tretej časti som robil malé úpravy v menších projektoch v redakčnom systéme "WordPress". Tejto časti sa v mojej bakalárskej práci venovať nebudem, keďže sa jednalo len o malé úpravy.

Prvá kapitola sa zaoberá spoločnosťou Moravio, s.r.o. a druhá technológiami, ktoré som využíval počas mojej odbornej praxe. Ďalšie kapitoly sa budú venovať jednotlivým úlohám a ich riešeniam.

1 Popis odborného zamerania firmy

Firma Moravio, s.r.o. sa zaoberá vývojom programových riešení na mieru, od webových stránok cez mobilné aplikácie až po informačné systémy a tiež "coodevelopmentom aplikácií" s inými firmami. Venuje sa aj online marketingu a dizajnu na mieru.

Pri vývoji softvéru na mieru sa firma zaoberá dizajnom na mieru podľa potreby a požiadaviek klienta. Začína sa analýzou potrieb a procesov, cez návrh, vývoj a uvedenie aplikácie do praxe u klienta. Na zefektívnenie práce používa firma rôzne nástroje ako sú napríklad "Jira"- nástroj pre efektívne riadenie vývoja softvéru, "GitLab" - nástroj pre správu zdrojových kódov a kontinuálnu integráciu nasadení na servery. Pre efektívnu internú komunikáciu firma využíva nástroj "Slack" a pre externú komunikáciu využíva "Basecamp".

Moravio sa zaoberá tvorbou vlastných webových aplikácií postavených prevažne na "PHP" ("Laravel") a "JavaScriptom" ("Vue.js"). Svoje aplikácie je firma schopná napojiť na služby tretích strán cez "API rozhranie".

Obrázok 1.1: Logo firmy Moravio, s.r.o.

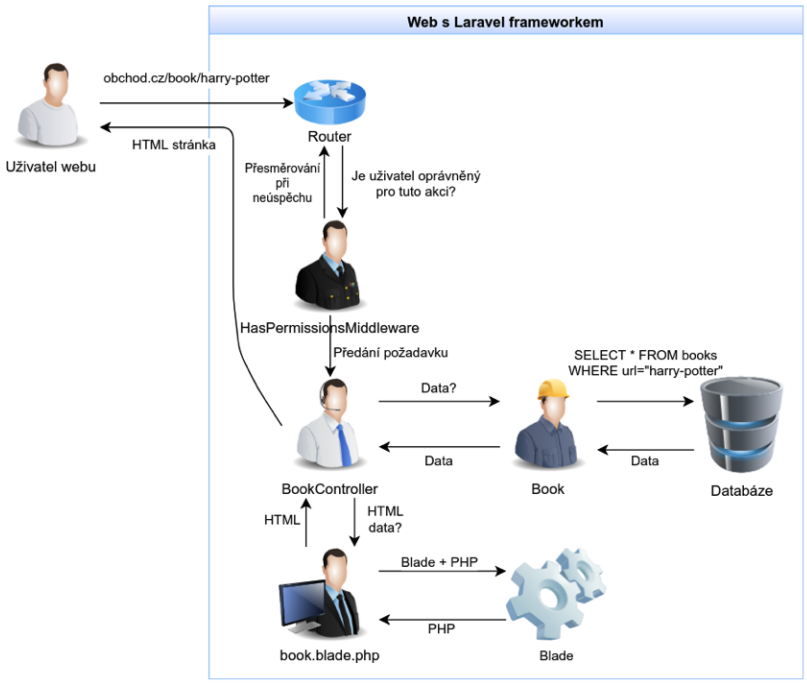
2 Použité technológie

Počas odbornej praxe som sa stretol s rôznymi technológiami, no prakticky všetky úlohy sa týkali programovacích jazykov PHP a JavaScript. Tiež som veľmi často pracoval s databázovým systémom "MySQL". V tejto kapitole stručne popisujem "frameworky" spomenutých programovacích jazykov a tiež niektoré ďalšie technológie, s ktorými som sa stretol pri práci.

2.1 Laravel (PHP)

Jedna z technológii, s ktorou som prišiel do styku, bol PHP framework Laravel. Tento framework funguje na architektúre MVC (Model-View-Controller). Úlohou MVC architektúry a teda aj tohoto frameworku je oddeliť logiku od rozhrania. Model reprezentuje databázu a logiku spojenú s dátami. View reprezentuje užívateľské rozhranie k aplikácii a Controller, ako už názov napovedá, riadi a spája tieto časti dohromady.

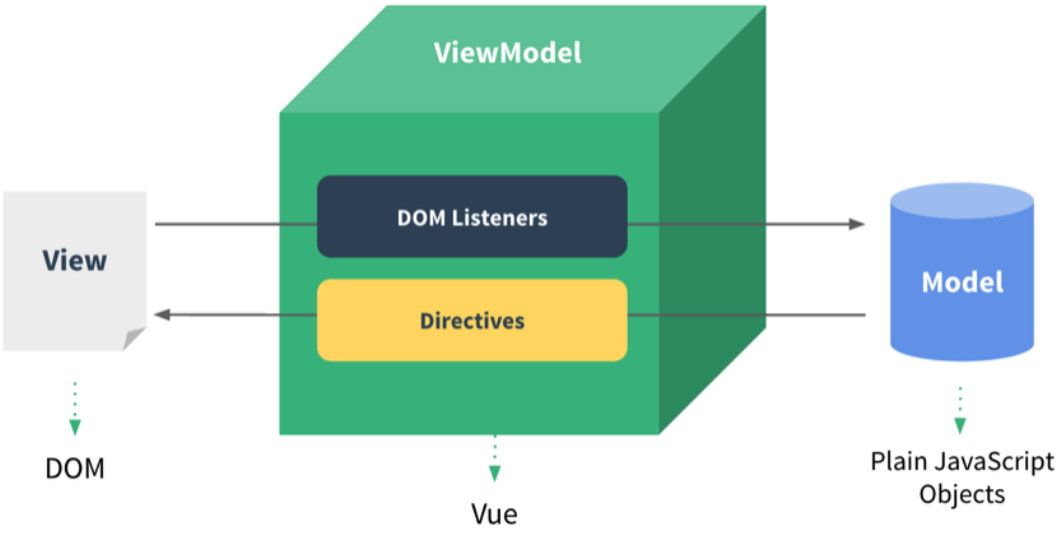
Laravel má mnoho vymožeností, ktoré uľahčujú prácu na väčších projektoch, a preto je to jeden z ideálnych kandidátov pre použitie na "backendovej" strane. Niektoré z týchto vymožeností sú napríklad: "Routing", Migrácie databázy, "Factories", "Eloquents", možnosť vytvárať vlastné CLI príkazy a mnoho ďalších. Na záver treba spomenúť, že Laravel funguje perfektne tam, kde sa počíta najmä s CRUD (Create-Read-Update-Delete) operáciami nad databázou.[2]



Obrázok 2.1: Životný cyklus webu bežiaceho na frameworku Laravel [1]

2.2 Vue.js (JavaScript)

Vue.js je jeden z top troch javascriptovských frameworkov, ktoré sa momentálne používajú (ďalšie dva sú "React" od Facebooku a "Angular" od Googlu). Pracuje s myšlienkou komponentov a ich znovu použiteľnosti. Vue.js sa používa pre tvorbu užívateľských rozhraní, takže dopĺňa "frontendovú" časť do už spomínaného backendového laravelu. Je teda primárne zameraný na View časť z MVC modelu. Typickým príkladom Vue.js komponentu môže byť napríklad vyhľadávacie políčko na stránke ("SearchBox"), ktoré vykoná nejakú akciu, po zadaní hľadaného výrazu užívateľom. Komponent následne zobrazí podobné vyhľadávacie výrazy a užívateľ si môže vybrať, ktorý výraz chce skutočne použiť (niečo podobné môžeme vidieť napríklad na youtube.com pri vyhľadávaní videa). [3]

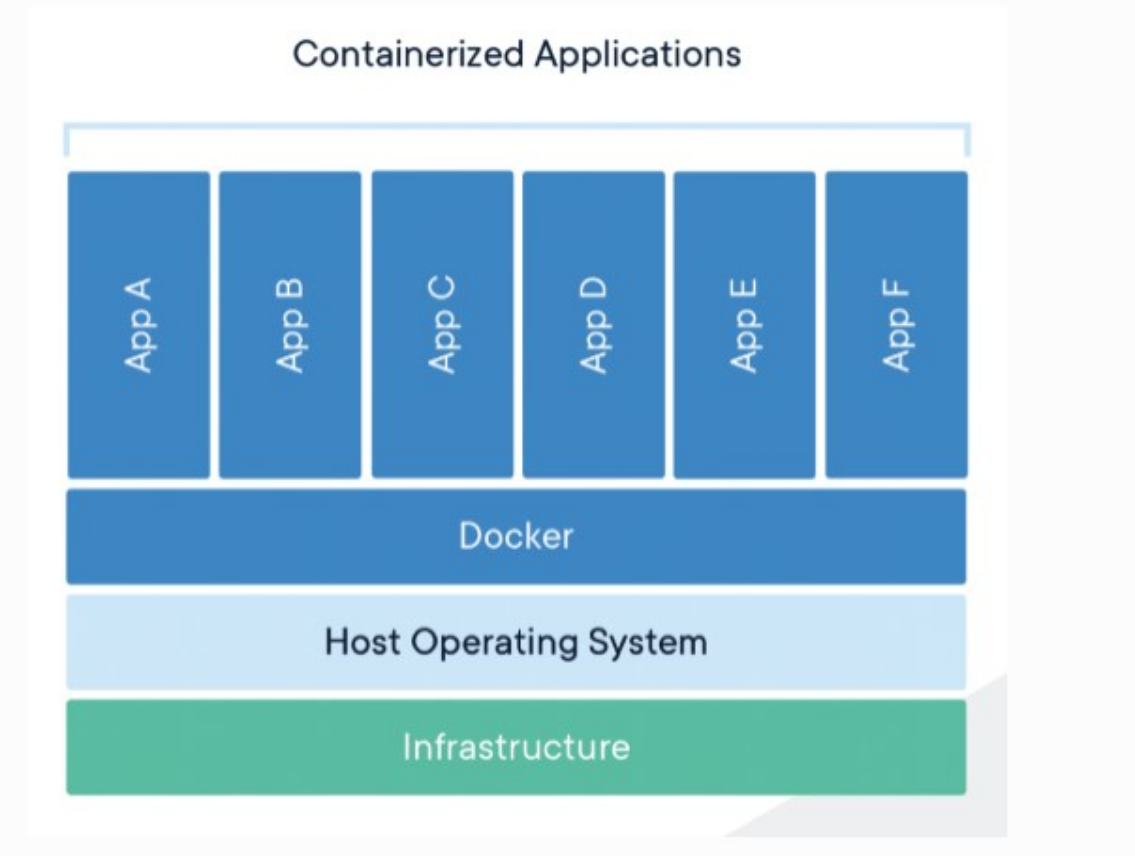


Obrázok 2.2: Princíp Vue.js (komunikácia model-view) [4]

2.3 Docker

"Docker" je technológia, ktorá okrem oddelenia jednotlivých aplikácií, ktoré bežia v jednotlivých kontajneroch, umožňuje tiež tvorbu aplikačných obrazov, ktoré je možné použiť v demonštračných a tiež produkčných prostrediach opakovane. To znamená, že Docker je skvelý nástroj pre automatizáciu časti procesu vývoja. Jednotlivé kontajnere by sa mali vzťahovať na jednu "mikroservisu". Zjednodušene povedané, Docker sa používa na zjednotenie vývojového prostredia naprieč všetkými OS. Nezáleží teda na akom operačnom systéme je produkt vyvíjaný, pretože služby, ako je napríklad MySQL, bežia nad úrovňou operačného systému, a teda rovnaký "docker image" vytvorí rovnaké prostredie. Počas praxe som spočiatku používal docker, no

neskôr, kvôli komplikáciám pri reinštalácii, som prešiel na "Homestead" od "Laravelu", ktorý je vlastne len iný typ virtualizácie a má veľmi podobný efekt ako Docker. [5] [6]



Obrázok 2.3: Princíp virtualizácie pomocou Docker kontajnerov [7]

2.4 MySQL

MySQL je relačný databázový systém, ktorý má na starosti uchovávať dáta v podobe tabuliek. Je multiplatformový a komunikuje pomocou SQL jazyka. Je veľmi populárny vďaka jeho jednoduchej implementácií. Pre jednoduchšiu prácu s MySQL sa používajú rôzni databázoví klienti (napríklad "phpMyAdmin" alebo "Sequel Pro"). Často sa používa v kombinácii: "Linux", PHP a MySQL. [8]

3 Informačný systém pre automatizáciu obchodu so stavebným odpadom

Tento projekt má slúžiť ako interný informačný systém pre klienta bežiaci na webovom rozhraní. Spoločnosť sa okrem iného zaoberá vývozom a recykláciou odpadového stavebného materiálu a presne pre tento účel bol tento systém navrhnutý.

Systém vie evidovať, upravovať a sledovať chemické analýzy materiálu, spravovať, priraďovať rôzne nákladové strediská a transporty, spravovať užívateľské roly, upravovať a vytvárať číselníky (napríklad typy odpadov) a nakoniec spravovať faktúry spoločnosti, týkajúce sa týchto procesov.

Mojou úlohou bolo prevažne vyvíjať časť s faktúrami, keďže na projekte sme pracovali v tíme a nie ako jednotlivci. Jednalo sa o niekoľko nadväzujúcich úloh, od vytvárania faktúr cez ich zobrazenie do "gridu" až po editáciu, ktorá sa opäť delila na niekoľko podčastí.

3.1 Konfigurácia projektu a oboznámenie sa s Laravelom

Mojou prvou úlohou bolo projekt, ktorý už bol v strednej fáze vývoja, rozbehnúť na mojom vývojovom prostredí. Projekt bežal na Docker kontajneroch, a teda bolo potrebné si projekt stiahnuť pomocou "Gitu" a spustiť si docker image pomocou príkazu *docker run [názov-image]*. Tento príkaz mi nastavil vývojové prostredie, ktoré bežalo pod Dockerom, presne s rovnakými parametrami, aké mali aj ostatní členovia tímu. Pod Dockerom bežalo hneď niekoľko služieb, ktoré mi zaisťovali, že som nemusel riešiť inštaláciu MySQL, alebo či mám rovnakú verziu PHP ako ostatní členovia tímu. Takisto už bol nainštalovaný framework Laravel a jediné čo bolo potrebné, bolo nastaviť si súbor *.env*, ktorý je v podstate srdcom Laravel projektu. V tomto súbore sa totiž nastavuje prístup do databázy, ktorú bolo treba vytvoriť pomocou nejakého klienta (v mojom prípade "Sequel Pro" pre "MacOS"). Nejednalo sa teda o produkčnú databázu, ale len databázu vytvorenú pre lokálny vývoj. Takisto bolo nutné pripojiť sa do kontajneru Dockera, v ktorom beží Laravel a vygenerovať "app key" pomocou príkazu *php artisan key:generate*. V súbore *.env* sa ďalej dá nastaviť mnoho vecí, ako napríklad meno našej aplikácie (*app name*), vypnutie/zapnutie "debuggera", testovacia databáza apod.

```
APP_NAME='App'
APP_ENV=local
APP_KEY=base64:1m2jUtDRppzPlydgK3ZtgXW/vhPStB8JNl2JioXrQqo=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://app.loc

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=test
DB_USERNAME=homestead
DB_PASSWORD=secret

DB_TEST_HOST=mysql
DB_TEST_PORT=3306
DB_TEST_DATABASE=app-test
DB_TEST_USERNAME=root
DB_TEST_PASSWORD=root
...
```

Obrázok 3.1: Príklad *.env súboru* pre Laravel projekt

V neposlednom rade bolo potrebné využiť vlastnosť, ktorú ponúka Laravel, a to spustiť migrácie mojej databázy. Ide o mechanizmus, ktorý dovoľuje vytvoriť databázu programovo a následne túto časť nasadzovať opakovane na ďalšiu a ďalšiu databázu. Ide teda o istý druh automatizácie. Tieto migrácie vytvorili tabuľky a ich stĺpčeky, ktoré bežia pod MySQL serverom.

No pre úplnú automatizáciu je potrebné mať tabuľky naplnené dátami. Aj toto má PHP framework Laravel vyriešené a umožňuje opäť programovo "zaseedovať", čiže naplniť databázu nejakými pseudo náhodnými dátami. Tvar týchto dát vieme špecifikovať pomocou tzv. "seederov", ktoré sa dajú v Laraveli vytvárať. Vo firme však tieto dva procesy, ktoré by normálne potrebovali dva príkazy na vykonanie, stlačili do jedného a migrácia a seedovanie teda prebehla po zadaní príkazu *make demo-db* cez terminál v zložke s projektom. Jedná sa o ďalšiu špecialitu Laravelu, a síce vytváranie vlastných príkazov do CLI.

Ako som už spomínal v úvode, tak Docker nie je jediný typ virtualizácie a existujú aj iné možnosti. Ja som neskôr prešiel na virtualizovaný Homestead server, ktorý je detailne popísaný na oficiálnych stránkach Laravelu [2]. Homestead je virtualizovaný linuxový server, bežiaci pomocou programu "VirtualBox". Pri mojej práci som sa totiž stretol s problémom zasielania zdvojenej hlavičky "http requestu" na server. Domnieval som sa, že je spôsobený práve nastaveným Docker kontajnerom, a preto som sa rozhodol využiť Homestead. Nakoniec sa však ukázalo, že problém bol v kóde, ale keďže sa mi na Homestead servery bežiacom cez VirtualBox pracovalo jednoduchšie, ostal som pri ňom už dokonca.

3.2 Vytvorenie "gridu" pre zoznam faktúr

Mojou nasledujúcou úlohou bolo vytvorenie podstránky pre zoznam faktúr spolu s príslušnými filtrami. Grid aj filtre bolo potrebné vytvoriť na serverovej strane a poslať ako celok do View, ktoré sa zobrazí užívateľovi. Takýto typ zobrazenia fungoval už na viacerých podstránkach, takže som využil už vytvorený grid, ktorý mal dve abstraktné funkcie *createFilter()* a *createGrid()*. Tie bolo potrebné implementovať.

Začal som funkciou *createFilter*. V prvom rade som potreboval do tabuľky dostať stĺpce, na ktoré sa vzťahujú jednotlivé filtre. Následne som jednotlivé filtre potreboval nastaviť na správny typ. Boli to tieto typy: "Select" (filtruje sa podľa jednej zvolenej hodnoty), "Multiselect" (filtruje sa podľa viacerých hodnôt zvolených naraz), "Text" (filtruje sa podľa textu, ktorý zadal užívateľ), Datepicker" (špeciálny prípad filtra, používal som naň javascriptovskú knižnicu na strane klienta). Selectu a Multiselectu bolo naviac treba pridať možnosti, ktoré sa po rozkliknutí zobrazia a možnosť, keď užívateľ nevyberie žiadnu konkrétnu položku (*NULL* možnosť).

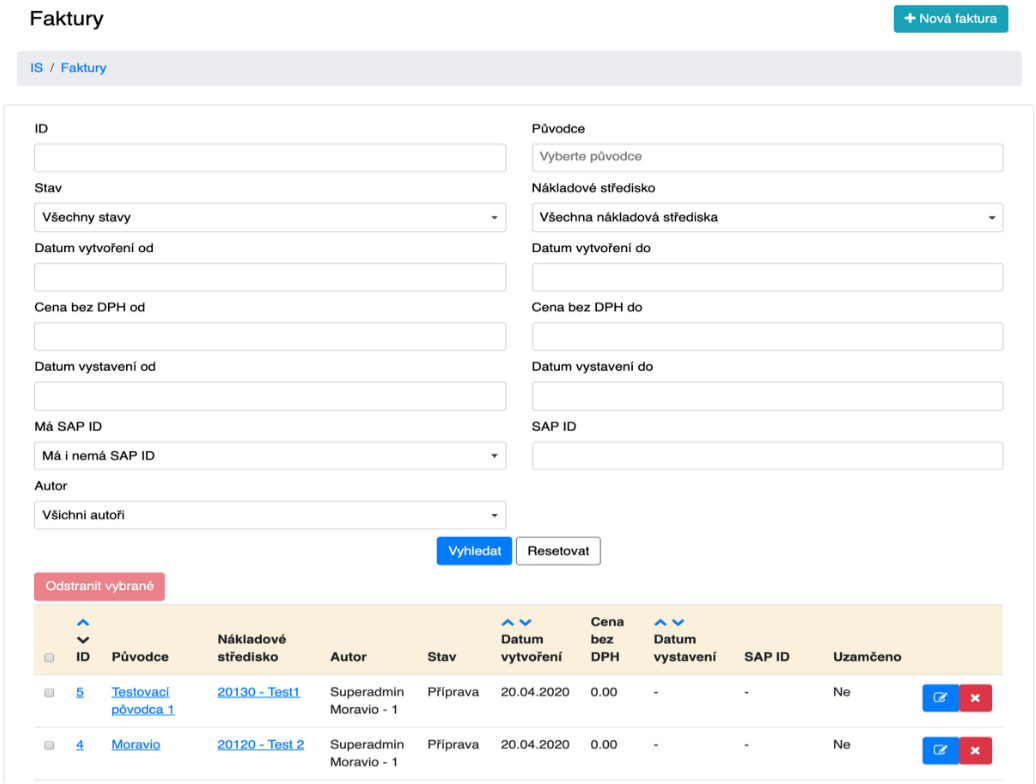
Druhá funkcia *createGrid* sa starala zasa o vytvorenie samotného zoznamu v tvare tabuľky. Tento grid som prepojil na filtre tak, aby dokázali filtrovať medzi dátami. Následne som postupne pridával stĺpčeky pre každé dáta, ktoré bolo potrebné zobraziť v tabuľke. Úplne prvý stĺpček obsahoval "checkbox" pre hromadné označovanie položiek gridu (napríklad pre hromadné mazanie). Posledný stĺpček zas obsahoval tlačidlá, ktoré začínali nejakú akciu po kliknutí. Konkrétne šlo o tlačidlá "*upravit*" a "*vymazat*". Tie sa užívateľovi zobrazili len v prípade, ak má na to právo. Pre niektoré stĺpčeky ako napríklad "*Id*", som musel ešte naviazať odkaz na podstránku danej faktúry, ktorá obsahovala faktúru vypísanú. Celý grid som následne vytvoril v Controlleri a poslal aj s filtrami do View, kde som ho následne zobrazil pomocou "HTML" a "CSS" frameworku "Bootsrap".

```
public function index()
{
    $this->authorize( ability: Ability::ABILITY_NAME__VIEW,  arguments: Invoice::class);
    $invoicesGrid = new InvoicesGrid();
    $filter = $invoicesGrid->getFilter();
    $grid = $invoicesGrid->getGrid();

    $this->breadcrumbService->addLevel(filter_route( name: 'invoices.index'), __( key: 'Faktury'));

    return view( view: 'invoices.index', compact( varname: 'filter',  _: 'grid'));
}
```

Obrázok 3.2: Funkcia Controlleru pre podstránku zobrazenia zoznamu faktúr

```php
protected function createFilter(): void
{
    $dataSource = Invoice::query()->select( columns: '*')->withGlobalScope(
        identifier: 'unique_name' . UserUniqueNameScope::class,
        new UserUniqueNameScope( idColumn: 'created_by_user_id', scopeName: 'userUniqName')
    )->withGlobalScope(
        identifier: 'subject_unique_name_' . SubjectUniqueNameScope::class,
        new SubjectUniqueNameScope( idColumn: 'subject_id', scopeName: 'subject_unique_name')
    );


    $this->filter = DataFilter::source($dataSource);
    $this->filter->attributes(['class' => 'form']);

    $this->filter->text( name: 'id', trans( key: 'ID'))
        ->scope(function (Builder $query, $value) {
            if ($value) {
                $query->withGlobalScope(
                    identifier: 'id_scope_' . ColumnEqualsScope::class,
                    new ColumnEqualsScope( column: 'id', $value)
                );
            }
            return $query;
        });

    $this->filter->multiselect('subject', trans( key: 'Původce'))
        ->attributes(['placeholder' => trans( key: 'Vyberte původce')])
        ->options($this->subjectRepository->createUniqueNameList());
```

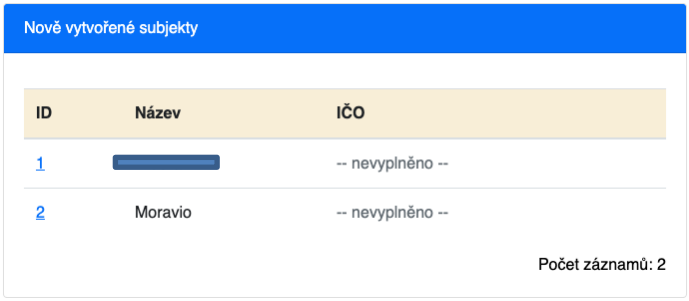
Obrázok 3.3: Časť funkcie vytvárania filtrov



Obrázok 3.4: Podstránka pre zobrazenie zoznamu faktúr

3.3 Vytvorenie "widgetu" na hlavnej stránke

Táto úloha bola zameraná na rozšírenie hlavnej stránky o "widget", ktorý mal zobraziť posledné vytvorené subjekty. Jednalo sa o jednoduchú úlohu, ktorá využívala podobný princíp ako úloha zobrazenia do gridu. Nebolo však potrebné riešiť filtre. Widget som následne musel naštýlovať pre responzívne zobrazenie na malých displejoch.



Obrázok 3.5: Widget pre zobrazenie nových vytvorených subjektov

3.4 Vytvorenie novej faktúry

Po výpise faktúr je nutné vedieť faktúry vytvárať. Na stránke s výpismi všetkých faktúr som teda pridal tlačidlo, po ktorého kliknutí sa zavolá "request" na server a ten podľa "*url*" zo súboru *web.php* zavolá tomu zodpovedajúci Controller, čiže presmeruje ma na podstránku pre vytvorenie faktúry. Podľa predchádzajúcich stránok, ktoré už boli hotové, som zhotovil "layout" stránky (hlavičku a drobčekovú navigáciu).

Na podstránke vytvorenia sú dva selektorové vstupy, pre určenie od akého pôvodcu faktúra je a z akého nákladového strediska pochádza. Oba selektory je potrebné naplniť zoznamom existujúcich pôvodcov alebo nákladových stredísk. Oba zoznamy som získal v Controlleri ako *kolekciu* a poslal som si ich na podstránku vytvárania faktúry. Pridal som tlačidlá na uloženie, ktoré odkazovali na metódu *store()* v Controlleri faktúr. Táto metóda má na starosti uloženie novej faktúry do databázy, a teda vykonanie transakcie. *Store* metóda musela najskôr autorizovať oprávnenia užívateľa k vykonania úkonu. Pre autorizáciu oprávnenia ponúka Laravel tzv. "middleware", čo je metóda, ktorá skontroluje oprávnenia pred úkonom. Implementuje sa jednoducho pomocou metódy *authorize()*, kde ako parameter posielame akciu (v našom prípade *create*) a modelu, na ktorý chceme akciu aplikovať. Ďalej som uložil request dáta, ktoré som poslal cez *StoreRequest* do Controlleru do premennej a pokúsil som sa ho pomocou *try-catch* uložiť. Na projekte bol používaný "Service layer pattern", takže všetka logika bolo presunutá do servisnej triedy prisluhujúceho modelu. Práve tam som vytvoril objekt, ktorý následne uložím do databázy s navolených dát a pridal som k nemu nejaké ďalšie atribúty, ako napríklad dátum vytvorenia alebo užívateľa, ktorý záznam vytvoril.

```php
public function create()
{
    $this->authorize( ability: Ability::ABILITY_NAME__CREATE,  arguments: Invoice::class);
    $subjects = $this->subjectRepository->createUniqueNameList();
    $costCenters = $this->costCenterRepository->createList();
    $this->breadcrumbService->addLevel(filter_route( name: 'invoices.index'), __( key: 'Faktury'));
    $this->breadcrumbService->addLevel(
        route( name: 'invoices.create'),
        __( key: 'Vytvoření faktury')
    );

    return view( view: 'invoices.create', compact( varname: 'subjects',  _: 'costCenters'));
}

/**
 * @param StoreRequest $request
 * @return \Illuminate\Http\RedirectResponse
 * @throws \Exception
 */
public function store(StoreRequest $request)
{
    $this->authorize( ability: Ability::ABILITY_NAME__CREATE,  arguments: Invoice::class);

    $invoiceData = $request->getData();

    try {
        DB::beginTransaction();

        $invoice = $this->invoiceService->create($invoiceData)->getModel();

        DB::commit();

        Session::flash('success', __( key: 'Uložení faktury proběhlo v pořádku.'));
    } catch (\Exception $e) {
        DB::rollBack();

        Sentry::captureException($e);

        Log::error( message: 'Vytvoření faktury se nezdařilo. ' . $e->getMessage());
        Session::flash('error', __( key: 'Vytvoření faktury se nezdařilo, zkuste to prosím znovu.'));

        return redirect()->back()->withInput();
    }

    return $this->formSubmitRedirect($request, $invoice);
}
```

Obrázok 3.6: Controller metóda *create*, a metóda *store* pre faktúry

3.5 Editácia faktúr

Nakoniec ešte bolo potrebné vyriešiť editáciu faktúr. Tá bola komplikovanejšia, keďže faktúry postupne nadobúdajú položky. Tie sa delia na 3 kategórie, a to príjemky, transporty a iné. Tieto položky musíme v editáciách vedieť pridávať a odoberať. Postup vytvorenia podstránky je taký istý ako v predchádzajúcej úlohe. Teda vytvorenie *url* vo *web.php* súbore a priradenie Controlleru k danej *url* a potom vytvorenie navigácie a drobčekovej navigácie. Táto podstránka sa skladá až z 5 častí a to: všeobecné údaje o editácii (pôvodca, stav, autor, dátum vystavenia…), odpadové príjemky, transporty, vlastné položky a kontrolné súčty cien.

V Controlleri v metóde *edit*, ktorá zodpovedá tejto podstránke, som musel získať všetky potrebné dáta, ktoré sa budú zobrazovať alebo spracovávať. Zahŕňalo to teda dáta o príjemkách, doprave (obe reprezentujú vlastný model, ktorý je previazaný väzbami na faktúru) a o samotných faktúrach. Veľká časť tejto úlohy sa vykonávala na strane klienta v javascriptovskom frameworku Vue.js. Tento framework som však nikdy predtým nepoužíval, takže som musel venovať veľa času tomu, aby som ho spoznal a naučil sa v ňom pracovať. V podstate som si vytvoril HTML súbor, do ktorého som pridával postupne vlastné *html tagy*, ktoré reprezentovali komponenty Vue.js. Tieto komponenty sú charakteristické tým, že majú oddelený kód od našej aplikácie (JavaScript, HTML a CSS), typicky v jednom súbore, ktorý reprezentuje daný komponent. Tento komponent je nutné potom zaregistrovať v súbore *App.js*, aby sme previazali vytvorený súbor komponentu so stránkou. Do každého komponentu som potom posielal dáta vo formáte "json", ktoré sme v danom komponente používali. Typické pre Vue.js je tiež to, že komponent sa často skladá z niekoľkých menších komponentov, ktoré spolu komunikujú. Komunikácia funguje na princípe *props-down*, *events-up*, čiže dáta-dole a udalosti-hore.

3.5.1 Vytváranie komponentov pre editáciu faktúr

Každý Vue.js komponent má životný cyklus, čomu zodpovedajú niektoré "*base metódy*" tohoto cyklu. Najpoužívanejšie metódy a teda aj stavy sú *created*, *mounted*, *updated* a *destroyed*. *Created* sa vykoná vtedy, keď sa náš komponent vytvorí, *mounted* vtedy, keď sa komponent pripojí na HTML stránku, *updated* vtedy, keď sa náš komponent nejako zmení a *destroyed* sa vykoná vtedy, keď náš komponent zanikne. Najskôr som teda vytvoril metódu *create*, v ktorej som "sparsoval" všetky dáta z "jsnu" a priradil ich do premenných v komponente. To zahŕňalo aj vytvorenie takzvaných *props*, ktoré Vue.js spracováva. Tie nám hovoria to, že aké dáta a v akom tvare posielame do príslušného komponentu. V metóde *mounted*, ktorá sa vykoná len raz za životný cyklus komponentu ako aj v metóde *created*, som nastavoval, či sa dáta majú uzamknúť a teda majú byť editovateľné alebo nie. Záležalo totiž od oprávnení, ktoré prihlásený užívateľ mal a podľa toho mohol upravovať jednotlivé vstupy s dátami. Okrem oprávnení záležalo aj na stave faktúry, a teda či bola v príprave, schválená alebo vyfakturovaná. Takto som vytvoril hlavný komponent, ktorý bol nadradený všetkým ostatným, a teda všetky potrebné dáta som spracoval a rozdistribuoval ďalej do podradených komponentov. V ňom prebiehala najpodstatnejšia logika ako serializácia dát, úprava formátu dát, mazanie položiek zo zoznamov, počítanie celkovej ceny, apod.

Posledná vec, ktorú som tu využíval, boli takzvané *Watchers*. Sú to metódy, ktoré sa vykonajú pri vpísaní dát do šablóny. Použil som to na výpis sumy v kontrolných súčtoch cien do faktúry. Časti pre výpis odpadov, vlastných položiek a dopravy boli viac menej identické tabuľky s položkami, ktoré bolo možné pridávať, odoberať a editovať.

Najjednoduchšie boli vlastné položky, keďže tie si užívateľ vyplnil úplne sám. Dopravy a odpady si užívateľ musel navoliť zo zoznamu existujúcich a mohol meniť už len cenu a množstvo. Všetky tabuľky obsahovali naviac kontrolný súčet.

```
<invoice-edit
    inline-template
    :show-detail="showDetail"
    :invoice="{{ $invoice->toJson() }}"
    :errors-json="{{ $errors->getBag('default')->toJson() }}"
    :custom-invoice-items-json="{{ collect(old('customInvoiceItems', $invoice->customInvoiceItems))->toJson() }}"
    :transports-json="{{ collect(old('transports', $invoice->transports->sortBy('asking_date')))->toJson() }}"
    :transport-types="{{ collect($transportTypes)->toJson() }}"
    :get-waste-type-price-url="'{{ route('invoices.get-waste-type-price') }}'"
    :projects="{{ $projects->toJson() }}"
    :waste-type-invoice-items-json="{{ collect(old('wtItems', $invoice->wasteTypeInvoiceItems))->toJson() }}"
    :can-edit="Boolean({{ Auth::user()->can(App\Models\Permissions\Ability::ABILITY_NAME__UPDATE, App\Models\Invoices\Invoice::class) }})"
>
```

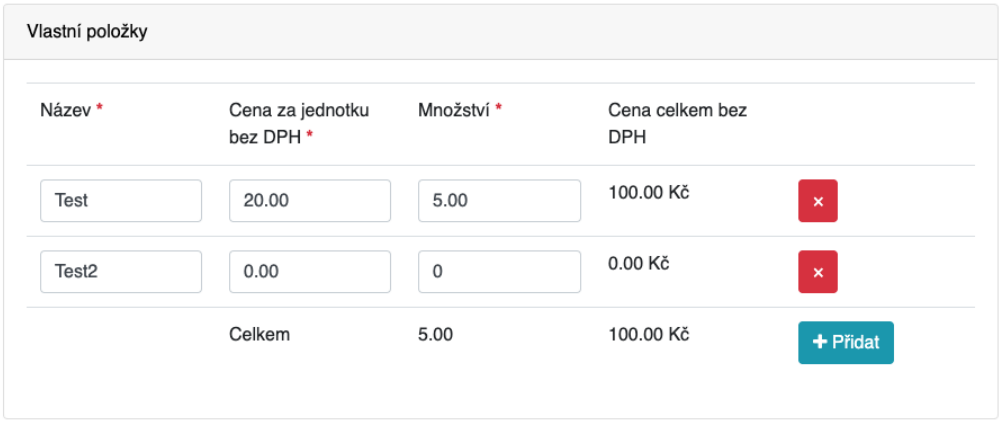
Obrázok 3.7: Použitie Vue.js komponentu v HTML šablóne

```
created() {
    this.errors = this.errorsJson;
    this.customItems = this.customInvoiceItemsJson;
    this.wtItems = this.wasteTypeInvoiceItemsJson;
    this.wtItems.forEach(item => {
        if (item.project_id === null) {
            item.project_id = 0;
            item.total_price = 0;
        }
    });
    this.transports = this.transportsJson;
},
```

Obrázok 3.8: JavaScript metóda *created* v *InvoiceEdit* komponente

```
watch: {
    customItems: {
        handler: function(newValue) {
            let totalPrice = 0;
            let totalAmount = 0;

            newValue.forEach(item => {
                totalPrice += +item.total_price;
                totalAmount += +item.amount;
            });

            this.customItemsPriceTotal = Number(totalPrice).toFixed( fractionDigits: 2);
            this.customItemsAmountTotal = Number(totalAmount).toFixed( fractionDigits: 2);
        },
        deep: true
    },
```

Obrázok 3.9: Vlastný *Watcher* v *InvoiceEdit* komponente



Obrázok 3.10: Komponent pre pridávanie a editáciu vlastných položiek na faktúre

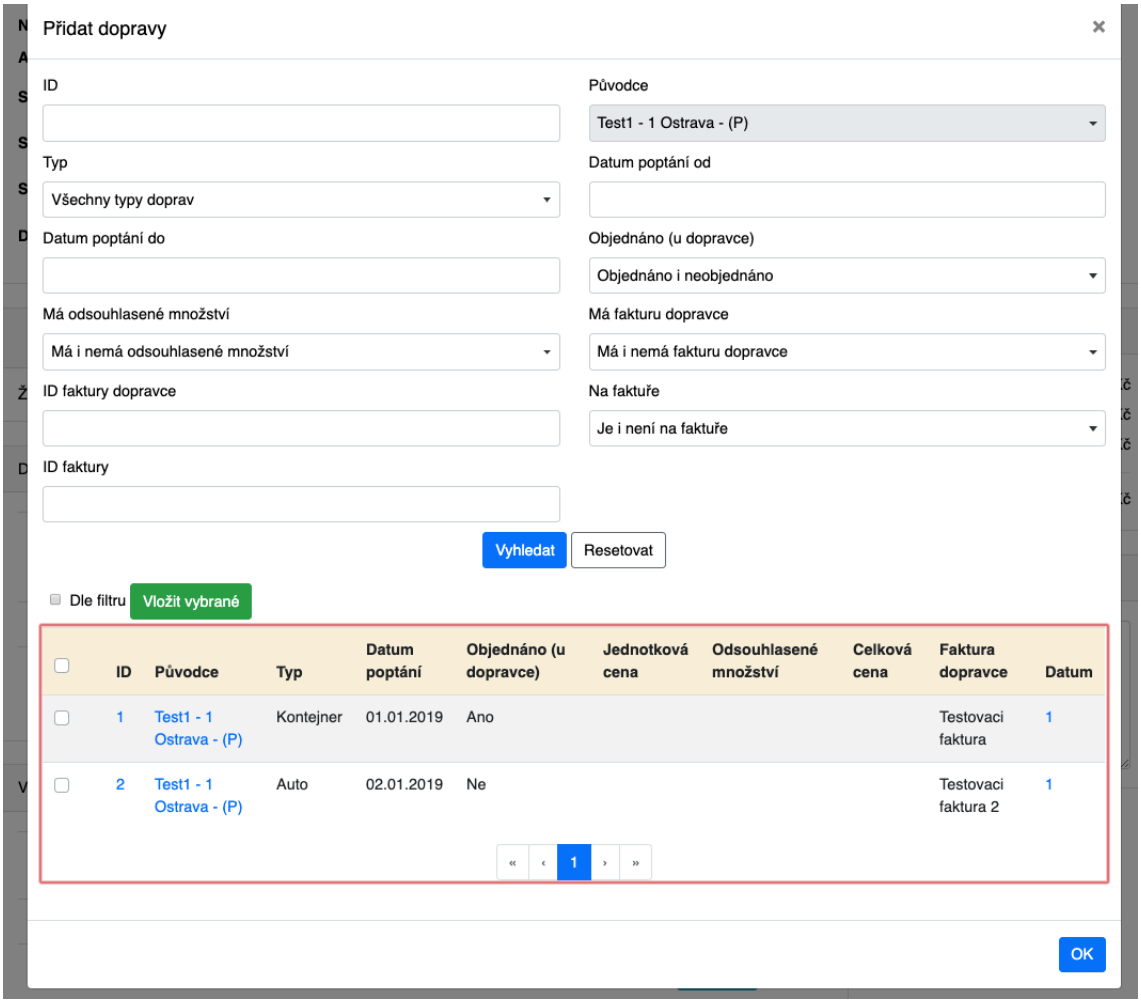
3.5.2 "Modaly" pre pridanie príjemky a transportu k faktúre

Výber príjemiek a transportov bol veľmi podobný. Prebiehali totiž v "modalovom" okne, čo je okno, ktoré sa zobrazí nad príslušnou podstránkou. Postup bol pri oboch takmer identický, len s menšou úpravou dát, takže nebudem popisovať oba postupy osobitne. Je potrebné spomenúť, že modalové okno sa skladalo z filtrov, tabuľky dát a akciových tlačidiel. Na vytvorenie modalu som využil Vue.js možnosť *inline templat*e, ktorá sa nevpisuje do JavaScript kódu, ale vpíše sa priamo do HTML šablóny a Vue.js si tieto súbory dokáže automaticky prepojiť. To mi umožnilo vytvoriť formuláre pomocou PHP značiek a kód bol teda efektívnejší, pretože som opäť mohol využiť princíp filtrov ako pri vytváraní gridu a filtrov pre zobrazenie zoznamu faktúr. Niektoré formulárové vstupy mali predvolené hodnoty, ktoré som do nich musel vložiť. Typy vstupov boli rôzne, niektoré boli obyčajné textové vstupy, niektoré Multiselecty, iné boli Selecty a tiež tu boli Datepickre. Zložitejšie boli práve Selecty a Multiselecty, na ktoré som používal externú knižnicu "Select2.js". Táto knižnica mi urýchlila prácu, keďže ponúkala hotové riešenie pre Selecty a Multiselecty, ktorým stačilo dodať zoznamy možností v požadovanom tvare.

Po otvorení modálu som teda pomocou API požiadal o dáta, ktoré som poslal naspäť, spracoval ich do požadovaných tvarov a zobrazil na príslušnom mieste v komponente. API v Laraveli funguje úplne rovnako ako obyčajné requesty cez *url*. Jediný rozdiel je v tom, že sú oddelené od seba. To znamená, že *url* sa nepíšu do súboru *web.php*, ale píšu sa do súboru *api.php* a taktiež Controllery sa píšu osobitne pre API routy. Tento Controller som nazval *InvoiceEditController* a skladal sa z minimálneho počtu metód, ktoré vracali transporty, príjemky a typy odpadov. Pretože vyhovujúcich záznamov mohlo byť mnoho, bolo potrebné riešiť stránkovanie. Šikovné riešenie a zároveň jednoduché na implementáciu ponúkal "BootstrapVue", a preto som sa ho rozhodol aj využiť. Toto riešenie funguje tak, že mu pošlem dáta, ktoré chcem zobraziť a nastavím parametre reprezentujúce počet riadkov na stránku. Po zvolení jednej alebo viac príjemiek sa tieto príjemky priradia k faktúre, no zatiaľ sa neuložia.

Môže však nastať situácia, keď je príjemka už priradená k inej faktúre, a vtedy bolo nutné nastaviť ďalšie modálne okno, ktoré je potrebné potvrdiť, pretože ide o neštandardnú situáciu. Je potrebné zmieniť sa ešte o špeciálnom prípade pridávania príjemiek a transportov pomocou checkboxu podľa filtra. V tomto prípade zavolám do "API" a pošlem tam dáta z filtrov. Server potom poskladá a vráti zoznam položiek, ktorý zodpovedá filtru a priradí to k faktúre. Ide teda o masové pridávanie položiek. Tiež je potrebné zobraziť a potvrdiť prípadné špecifické operácie, ako už spomínané priradenie príjemky z inej faktúry, do faktúry práve zvolenej.

Nakoniec som pridal ešte jeden typ modálneho okna, a to na zobrazenie detailov už priradenej príjemky alebo transportu. Ide len o výpis informácii k danej položke, bez nutnosti presmerovania na podstránku danej príjemky alebo transportu.



Obrázok 3.11: Modálne okno pre pridanie transportu v editácii faktúry

3.5.3 Uloženie editovanej faktúry

Editované faktúry som musel uložiť do databázy. Musel som teda celú stránku editácii obaliť do *formulárového tagu*. Pridal som tlačidlá na ukladanie alebo zrušenie editácie a nastavil som všetky názvy vstupov, aby zodpovedali databázovému zápisu. To bolo dôležité pri overovaní správneho formátu dát. Nemusel som teda riešiť "preuloženie" dát do iných premenných, než aké mi boli poslané. Tlačidlo uloženia som teda naviazal na metódu *update* v Controllery.

Táto metóda má dva parametre - zmodifikovaný *updateRequest* a "*Id*" editovanej faktúry. *UpdateRequest* je upravená trieda, ktorá dedí z *baseFormRequest* triedy. Ide o triedu, ktorá ma na starosti spracovanie dát, ktoré sme poslali pomocou nášho formulára. Formulár nám umožňuje vytvárať si pravidlá, ktoré majú skontrolovať, či sú v správnom tvare, či majú požadovanú dĺžku alebo či sú vyplnené povinné polia. Taktiež táto trieda umožňuje vytvárať správy, ktoré sa

zobrazia v prípade, že niektorí zo vstupov nespĺňa niektoré z pravidiel. No a nakoniec umožňuje pridať metódu, ktorá sa stará o to, v akom tvare nám budú dáta po overení vrátené.

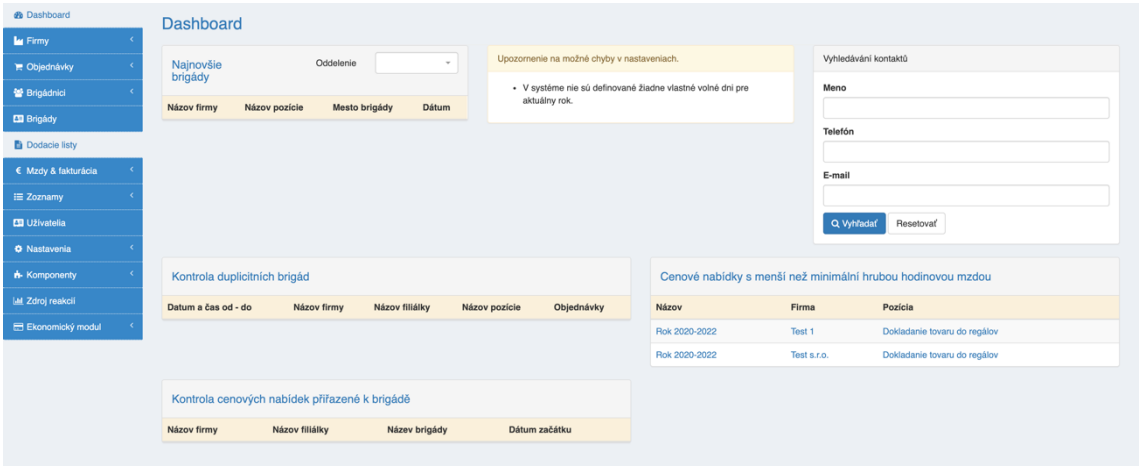
Konkrétne v mojom prípade som mal vo formulári položku *sap id*, ktorú mohli editovať iba oprávnení užívatelia a to iba v niektorých *stavoch* faktúry. To znamená, že dáta mohli byť prepísané na strane užívateľa, aj keď mal zablokované niektoré polia formulára, a teda vznikla by tu náchylnosť na nechcenú modifikáciu dát. Práve preto som musel tieto dáta z formulára vymazať pre neoprávnené osoby a uložiť zmeny bez nich. Následne som sa v bloku *try - catch* snažil dáta uložiť do databázy. V prípade, že sa toto nepodarilo, mali sa dáta poslať späť na podstránku editovanej faktúry a zostať predvyplnené. To Laravel umožňuje pomocou metódy *withInputs()*. Potom bolo už len potrebné, navrátene dáta zobraziť spolu s chybnými hláškami k danému vstupu, a to len v prípade, že nejaká chyba na vstupe bola. Naopak, v prípade úspechu, sa zobrazila hláška o potvrdení uloženia faktúry do databázy.

```php
public function update(UpdateRequest $request, Invoice $invoice)
{
    $this->authorize( ability: Ability::ABILITY_NAME__VIEW,  arguments: Invoice::class);

    $invoiceData = $request->getData();

    try {
        DB::beginTransaction();

        $this->invoiceService->setModel($invoice)->update($invoiceData);

        DB::commit();

        Session::flash('success', __( key: 'Uložení faktury proběhlo v pořádku.'));
    } catch (ModelCanNotBeUpdated $e) {
        DB::rollBack();

        Session::flash('error', $e->getMessage());

        return redirect()->back();
    } catch (\Exception $e) {
        DB::rollBack();

        Sentry::captureException($e);
        Log::error($e);
        Log::error( message: 'Uložení faktury se nezdařilo. ' . $e->getMessage());
        Session::flash('error', __( key: 'Uložení faktury se nezdařilo, zkuste to prosím znovu.'));

        return redirect()->back()->withInput();
    }

    return $this->formSubmitRedirect($request, $invoice);
}
```

Obrázok 3.12: Editácia faktúr - *update* metóda

4 Informačný systém pre firmu zaoberajúcu sa sprostredkovaním brigád

Môj druhý projekt, na ktorom som sa podieľal, bol informačný systém, ktorý slúži na správu brigád a brigádnikov. Firma eviduje brigády od tretích strán a ponúka ich verejnosti. Informačný systém teda tento proces z veľkej časti automatizuje. V tomto prípade sa jednalo o projekt, ktorý už bol funkčný, ale firma mala záujem ho rozšíriť o modul, ktorý by mal za úlohu zobraziť informácie ekonomického charakteru vo zvolených intervaloch. Bolo potrebné rozšíriť už existujúcu databázu o nové dáta a tiež bolo potrebné zapracovať zobrazenie týchto dát do grafov podľa zvolených parametrov.



Obrázok 4.1: Hlavný panel brigádnického informačného systému

4.1 Konfigurácia projektu

Konfigurácia projektu bola podobná ako v predchádzajúcom projekte, keďže sa v ňom používali totožné technológie. Súčasťou tejto konfigurácie bolo aj školenie, ako informačný systém funguje, pretože sa jednalo o rozsiahlejší informačný systém, ktorý už bol nasadený v produktívnom režime.

4.2 Úprava databázy

Vzhľadom na to, že sa jednalo o úplne nový modul do systému, bolo nutné rozšíriť databázu o nové atribúty. Ako už bolo v predošlých častiach mojej práce spomenuté, v Laraveli sa na prácu s databázovou vrstvou používajú migrácie. Mal som za úlohu pridať do tabuliek dodacích listov, objednaných prác a pracovných ponúk nové atribúty, ako napríklad profit, objednané hodiny, maximálny počet hodín apod. Trieda každej migrácie obsahuje dve metódy - *up* a *down*. V metóde *up* som popísal logiku, ktorá sa vykoná pri vytváraní databázy a v metóde *down* logiku, ktorá sa vykoná pri jej zhodení. Atribúty sa pridávajú pomerne jednoducho. Zvolil som si tabuľku a pomocou metód som vytvoril nové atribúty, ktoré som nastavil tak, ako bolo potrebné. Pridal som im poznámku, čo daný atribút reprezentuje, nastavil predvolené hodnoty a

určil, na aké miesto v tabuľke sa majú zapísať (poradie). V metóde *down* som tieto vytvorené atribúty *droppoval* (zhadzoval).

```php
public function up()
{
    Schema::table('delivery_notes', function (Blueprint $table) {

        $table->string( column: 'ordered_hours_count')->after('count_notify_worker_sended')->comment('Počet objednaných hodin');
        $table->string( column: 'agreed_hours_count')->after('ordered_hours_count')->comment('Domluvený počet hodin');
        $table->string( column: 'maximum_margin')->after('agreed_hours_count')->comment('Maximální marže')->default(0);

        $this->addCreatedAndUpdatedBy($table);
        $table->string( column: 'profit')->after('maximum_margin')->comment('Zisk')->nullable();
        $table->index( columns: 'profit');
    });

    Schema::table('order_works', function (Blueprint $table) {
        $table->string( column: 'hours_capacity')->after('total_missing')->comment('Hodinová kapacita brigády');
        $table->string( column: 'maximum_margin')->after('hours_capacity')->comment('Maximální marže')->default(0);
        $table->string( column: 'profit')->after('maximum_margin')->comment('Zisk')->nullable();
        $table->index( columns: 'profit');
    });

    Schema::table('job_offers', function (Blueprint $table) {
        $table->string( column: 'maximum_margin')->after('name')->comment('Maximální marže')->default(0);
    });
}
```

Obrázok 4.2: Príklad triedy zodpovednej za migráciu databázy

4.3 CLI príkazy pre upravenie databázy a prepočítanie dát

Laravel umožňuje vytvárať vlastné príkazy do príkazového riadku, ktoré sú spustené na strane servera po zadaní príkazu *php artisan [príkaz]*. Úlohou bolo vytvoriť dva príkazy.

Prvý príkaz mal prepočítať počet autorov dodacích listov z *logov*. Dôvodom bolo to, že dodacie listy nemali uloženého svojho autora, keďže databáza nebola navrhovaná pre tento nový modul. Niektoré potrebné dáta teda chýbali a bolo ich treba doplniť. Problémom bolo, že ostrá databáza mala tisícky až stovky tisíc záznamov. Výpočet by teda pri nesprávnom postupe mohol trvať veľmi dlho. Zvolil som postup, pri ktorom som si načítaval dáta na spracovanie postupne (po takzvaných "chunkoch"). Tieto dáta som spracoval a uložil, a potom som cyklicky načítaval nové dáta po častiach až do konca. Nakoniec som výsledok "úspechu" alebo "neúspechu" vypísal do CLI.

Druhý príkaz, na ktorom som pracoval, mal dopočítať hodnotu pre nový atribút, ktorý som pridal do databázy. Tento atribút bol profit dodacích listov alebo objednaných prác. Postup bol veľmi podobný, ale logika bola trošku zložitejšia. Preto bolo na mieste vytvoriť novú servis triedu a do nej metódu na prepočet tohoto profitu. Tu som si už zaznamenával aj pokrok, aby bol užívateľ informovaný v priebehu prepočtu údajov. Tiež som musel odpojiť "*eventy*" (udalosti) naviazané na model, pretože pri takejto manipulácii s dátami dochádzalo k ich nežiaducemu vyvolaniu. Následne som opäť po chunkoch spracovával dáta a nakoniec som výsledok vypísal do CLI.

```php
public function recountProfitability(Model $model, int $skip) : void
{
    $lastId = 0;
    $progress = 0;
    $count = $model::where( column: 'id',  operator: '>=', $skip)->count();
    $dispatcher = $model::getEventDispatcher();
    $model::unsetEventDispatcher();

    if ($model instanceof DeliveryNote) {
        $function = 'recountDeliveryNoteProfit';
    } elseif ($model instanceof OrderWork) {
        $function = 'recountOrderWorkProfit';
    } else {
        throw new \Exception(sprintf( format: 'Cannot recount profit of %s.', class_basename($model)));
    }

    try {
        $model::where( column: 'id',  operator: '>=', $skip)
            ->chunk( count: self::CHUNK_SIZE, function ($chunk) use (
                $count,
                $function,
                $lastId,
                $model,
                &$progress,
                $skip
            ) {
                \DB::beginTransaction();

                foreach ($chunk as $chunkPart) {
                    $this->$functionX($chunkPart);
                    $progress++;
                    $lastId = $chunkPart->id;
                }

                \DB::commit();
                \Log::info( message: 'Calculated profit for ' . class_basename($model)
                    . ': ' . $progress . '/' . $count . ' (Last ID: ' . $lastId . ')');
            });
    } catch (\Exception $e) {
        echo $e->getMessage();
        $model::setEventDispatcher($dispatcher);
        \DB::rollBack();
        throw new \Exception( message: 'Error during recounting of delivery notes profit');
    }

    $model::setEventDispatcher($dispatcher);
}
```

Obrázok 4.3: Metóda pre prepočet profitu

_134548_update_economic_module.php | ComputeDeliveryNotesAuthors.php | AbsencesAndLostsDataService.php

```php
public function handle()
{
    $CHUNK_LENGTH = 500;
    try {
        \Config::set('disableUpdatedByUser', true);
        $this->counter = 0;

        //disable index updates to increase import speed
        \DB::raw( value: 'SET FOREIGN_KEY_CHECKS = 0');
        \DB::raw( value: 'SET UNIQUE_CHECKS = 0');
        \DB::raw( value: 'SET AUTOCOMMIT = 0');
        \DB::raw( value: 'ALTER TABLE `delivery_notes` DISABLE KEYS');


        DB::table( table: 'activity_log_entries')
        ->where( column: 'action', operator: '=', value: 'created')
        ->where( column: 'entity_type', operator: '=', value: 'App\Models\DeliveryNotes\DeliveryNote')
        ->where( column: 'user_id', operator: '<>', value: null)
        ->orderBy( column: 'id')
        ->chunk($CHUNK_LENGTH, function ($logs) {
            \DB::beginTransaction();
            foreach ($logs as $log) {
                $delivery_note_id  = $log->entity_id;
                $autor_id = $log->user_id ;

                $deliverynote = DeliveryNote::find($delivery_note_id);
                if ($deliverynote !== null) {
                    $deliverynote->update([
                        'created_by_user_id' => $autor_id,
                        'updated_by_user_id' => $autor_id,
                    ]);
                    $this->counter += 1;
                }
            }
            \DB::commit();
        });

        //reenable index updates
        \DB::raw( value: 'ALTER TABLE `delivery_notes` ENABLE KEYS');
        \DB::raw( value: 'SET FOREIGN_KEY_CHECKS = 1');
        \DB::raw( value: 'SET UNIQUE_CHECKS = 1');

        $this->output->success('Celkem bylo updatováno ' . $this->counter . ' záznamú.');
        \Config::offsetUnset( key: 'disableUpdatedByUser');
    } catch (\Exception $e) {
        \DB::rollBack();
        $this->output->success('Pri prepočítavaní autorov dodacích listov došlo k chybe.');
    }
}
```

Obrázok 4.4: CLI príkaz - prepočet autorov dodacích listov

4.4 Routing a hlavičková navigácia pre podstránky v novom module

Mojou úlohou bolo pripraviť všetky podstránky, nastaviť routing (smerovanie), vytvoriť v navigácii tlačidlá k novým podstránkam a vyrobiť drobčekové navigácie pre každú podstránku. Tieto podstránky boli štyri - ziskovosť, marža, obsadenosť a absencie so stratami. Každá podstránka mala vyzerať rovnako, ale zobrazovať iné dáta.

Pridal som teda do navigácie tzv. vysúvaciu navigáciu pre ekonomický modul a pod ňou som pridal všetky štyri nové podstránky. Pre každú podstránku som potom vytvoril novú cestu vo *web.php* a s ňou nový Controller pre každú podstránku. Potom som vytvoril v každom Controllery drobčekovú navigáciu a nastavil Controller tak, aby vrátil príslušnú podstránku. Každú podstránku som musel vytvoriť pomocou rozšírenia hlavnej šablóny, keďže bolo potrebné, aby layout stránky ostal rovnaký a meniť sa mala len určitá časť stránky. Každá podstránka však obsahovala ešte jedno ďalšie menu, ktoré menilo zobrazenie (napríklad ziskovosti) pre firmu, zamestnancov alebo zamestnanca. Toto menu som dorobil tiež na všetky podstránky. Teraz boli všetky nové podstránky pripravené pre ďalší vývoj.

4.5 Trieda servisy pre rozdelenia roka po intervaloch (týždne, dni, mesiace)

Na každej podstránke boli potom dorobené filtre. Jedným z filtrov bola *členitosť* a jej úlohou bolo určiť, po akých úsekoch chceme dáta filtrovať. Možné úseky boli - dni, týždne, mesiace a roky. Preto bolo nutné vytvoriť novú "servisu", ktorá sa o toto členenie starala a navrátila požadované úseky. Pre každú *členitosť* som teda vytvoril triedu, ktorá implementovala "interface", keďže sa menila len logika toho ako sa interval poskladá.

V prvom rade bolo potrebné skontrolovať, či je interval správny. Príkladom zle zvoleného intervalu by bolo to, že konečný dátum je nižší ako počiatočný dátum. V takom prípade sa vyhodila *exception* (výnimka) a užívateľ musel navoliť iný interval. Na prácu s dátumami som použil externú knižnicu "Carbon".

Pre spracovanie členitosti po dňoch som musel kontrolovať, či zvolený interval je v priestupnom roku. V takom prípade má rok totiž 366 dní, a to bolo nutné zohľadniť.

Horšie to bolo pri *členitosti* po týždňoch, kde som zistil, že prvý alebo posledný týždeň zo zvoleného roku môžu presahovať do iného roku. To do akého roku tieto týždne patria závisí od toho, že v ktorom roku má viac dní z týždňa. Keď napríklad v roku XXX1 pripadá prvý deň na piatok, tak tento týždeň bude patriť ešte do roku XXX0.

Vykonať *členitosť* po mesiacoch a rokoch bolo jednoduchšie. Nemusel som tam riešiť žiadne špeciálne situácie.

Tieto metódy mi teda vrátili objekt, ktorý obsahoval intervaly od-do rozdelené podľa zvolenej členitosti. Tieto intervaly som potom použil pre zobrazenie údajov do grafu v ďalšej úlohe.

_134548_update_economic_module.php | WeekIntervalStatisticService.php | ComputeDeliveryNotesAuthors.php | AbsencesAndLos

```php
     * @return Collection|mixed
     * @throws InvalidIntervalException
     */
    public function getIntervals(Carbon $startOfInterval, Carbon $endOfInterval, $sinceYear = null)
    {
        if (!$this->isIntervalValid($startOfInterval->copy(), $endOfInterval->copy())) {
            throw new InvalidIntervalException( message: 'Vybraný interval je nesprávny');
        }

        $weeks = collect();
        $toYear = Carbon::today()->year;
        $sinceYear = $sinceYear ?? $startOfInterval->year;

        $years = range($sinceYear, $toYear);

        /** @var WeekInterval[] $referenceYearIntervals */
        $referenceYearIntervals = collect();
        $referenceYear = $startOfInterval->copy()->endOfWeek()->year;
        $endDate = $endOfInterval->copy()->endOfWeek();
        $date = $startOfInterval->copy()->startOfWeek();

        while ($date <= $endDate) {
            $startOfWeek = $date->copy();
            $endOfWeek = $date->copy()->endOfWeek();

            $interval = new WeekInterval($startOfWeek, $endOfWeek, $startOfWeek->weekOfYear);
            $referenceYearIntervals->push($interval);

            $date->addWeek();
        }
        $weeks->put($referenceYear, $referenceYearIntervals);

        foreach ($years as $year) {
            // Skip reference year
            if ($years === $referenceYear) {
                break;
            }

            $yearIntervals = collect();
            foreach ($referenceYearIntervals as $referenceYearInterval) {
                $weekNumber = $referenceYearInterval->getWeekNumberInYear();

                $startOfWeek = new Carbon();
                $startOfWeek->setISODate($year, $weekNumber);
                $endOfWeek = $startOfWeek->copy()->endOfWeek();

                $interval = new WeekInterval($startOfWeek, $endOfWeek, $weekNumber);
                $yearIntervals->push($interval);
            }

            $weeks->put($year, $yearIntervals);
        }

        return $weeks;
    }
```

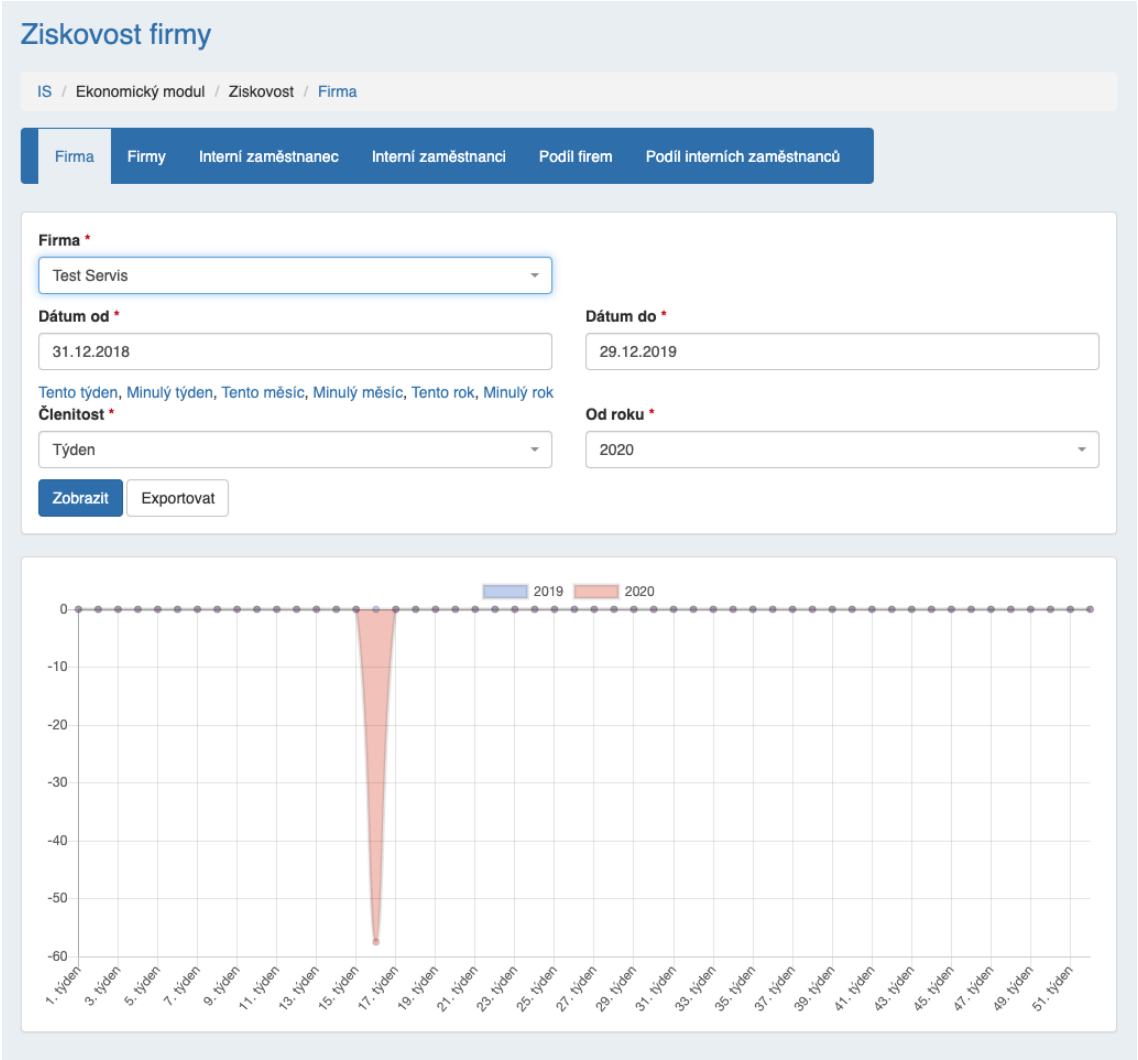
Obrázok 4.5: Metóda na prepočet týždňových intervalov

4.6 Vykreslenie dát do grafov pomocou Chart.js

Mojou poslednou úlohou bolo zobraziť vyfiltrované údaje do grafu. Na vykreslenie grafu som použil JavaScriptovskú knižnicu "Chart.js". Táto knižnica umožňuje skvelú prácu s grafmi a umožňuje nastaviť si grafy do ľubovoľného tvaru. Typ grafov sa líšil pre každú podstránku, no typ grafov určovalo len nastavenie komponentu grafu a tvar dát, ktorý do komponentu pošleme. Do Controlleru som teda dopracoval logiku, ktorá kontrolovala, či už boli filtre vyplnené a ak áno, tak naplnili graf dátami vo zvolených intervaloch. Tento postup som opakoval v každom Controllery s malými zmenami hlavne v tvaroch dát.

```
valStatisticService.php | AbsencesAndLostsController.php | AbsencesAndLostsDataService.php
 * @param LostHoursRequest $request
 * @return \Illuminate\Contracts\View\Factory|\Illuminate\Http\RedirectResponse|\Illuminate\Routing\Redirector|\Illum
 * @throws \Modules\Economic\Exceptions\InvalidIntervalException
 * @throws \Modules\Economic\Exceptions\InvalidSegmentationException
 */
public function lostHours(LostHoursRequest $request)
{
    if ($this->intervalNeedsFix($request)) {
        return  $this->getRedirectWithFixedInterval($request);
    }
    $request->saveFilterRoute();

    $chartData = [];
    if ($request->isSubmitted()) {
        $chartData = $this->absencesAndLostsDataService->getLostHoursChartData(
            $request->getUserId(),
            $request->getCompanyId(),
            $request->getDateFrom(),
            $request->getDateTo(),
            $request->getSegmentation()
        );

        if ($request->isSubmittedExport()) {
            $template = resolve( name: AbsencesAndLostsChartTemplate::class);
            $template->setData($chartData);
            $result = $template->getTemplate();
            return response()->download($result,  name: self::LOST_HOURS_EXPORT_FILENAME);
        }
    }

    $segmentationSelectOptions = $this->getSegmentationSelectOptions();
    $fromYearSelectOptions = $this->getYearFromSelectOptions();
    $companySelectOptions = $this->getCompaniesSelectOptions();
    $userSelectOptions = $this->userRepository->createListForSelect();

    $this->breadcrumbService
        ->addLevel(filter_route( name: 'economic.absences-and-losts'), __( key: 'Ztracené hodiny'));

    return view(
         view: 'economic::absencesAndLosts/lostHours',
        compact(
             varname: 'segmentationSelectOptions',
            _: 'fromYearSelectOptions',
            'companySelectOptions',
            'userSelectOptions',
            'chartData'
        )
    );
}
```

Obrázok 4.6: Conroller metóda pre zobrazenie stratených hodín v grafe



Obrázok 4.7: Podstránka pre zobrazenie ziskovosti firmy

5 Teoretické a praktické vedomosti a zručnosti získané v priebehu štúdia a uplatnené študentom v priebehu odbornej praxe.

V priebehu mojej praxe som využil mnoho vedomostí, ktoré som nadobudol počas štúdia. Vhodne som si zvolil niektoré dobrovoľné predmety, ktoré boli taktiež cenným pomocníkom pri vypracovávaní zadaných úloh.

Najviac som využil vedomosti z predmetu "Tvorba aplikácií pre mobilné zariadenia I.", pretože som veľmi často pracoval s programovacím jazykom JavaScript alebo s jeho knižnicami. Tak isto všetky predmety, ktoré sa zaoberali programovaním, boli veľmi užitočné, a to najmä tie, ktoré pracovali s objektovo orientovaným programovaním.

Pri vypracovaní niektorých zložitejších úloh som pri výpočtoch používal vedomosti z predmetov "Algoritmy I. a II." spolu s predmetom "Úvod do teoretickej informatiky".

Počas môjho štúdia som si navyše zvolil predmety ako "Úvod do databázových technológií" a "Vývoj internetových aplikácií". Aj vedomosti z týchto predmetov sa mi veľmi hodili a využil som ich. Vedomosti z predmetu "Úvod do databázových technológií" mi obzvlášť pomohli pri práci s databázami a predmet "Vývoj internetových aplikácií" podstatne rozšíril moje vedomosti o JavaScripte, ktorých základy som získal počas štúdia predmetu "Tvorba aplikácií pre mobilne zariadenia I."

V neposlednom rade by som chcel vyzdvihnúť predmet "Angličtina", bez ktorého by dnes nemohol fungovať asi žiaden programátor. Používal som ho na dennej báze, či už na komunikáciu, písanie kódov alebo pri čítaní dokumentácie na internete.

6 Vedomosti alebo zručnosti, ktoré študentovi chýbali v priebehu odbornej praxe

Na tomto mieste by som rád spomenul aj vedomosti, ktoré mi naopak počas výkonu praxe trošku chýbali. Určite by som privítal viac vedomostí z technológie "Node.js" a hlbšiu znalosť nejakého frameworku, či už na strane klienta (JavaScript) alebo na strane servera (PHP). Tieto frameworky majú často hotové riešenia na mnoho opakujúcich sa úloh a zároveň sú tieto riešenia veľmi dobre spracované po technickej stránke.

Spočiatku mi tiež chýbala znalosť linuxových serverov. Predmet zameraný na linuxové servery som síce mal, ale až v poslednom semestri už počas praxe, čo pre moje potreby vyplývajúce z tejto práce nebolo úplne ideálne. Chýbali mi tiež hlbšie poznatky o niektorých vývojárskych nástrojoch, ako napríklad Git, Jira alebo Confluence.

7 Záver

Vďaka tejto odbornej praxi som si rozšíril vedomosti v oblasti webových technológií a bližšie pochopil proces vývoja softvéru. Vyskúšal som si prácu v tíme, ktorá je podstatne odlišná od práce jednotlivca. Naučil som sa pracovať s novými technológiami a nástrojmi a tiež som dostal veľmi veľa tipov ako programovať efektívnejšie.

Nové nadobudnuté vedomosti a skúsenosti však neboli iba technického charakteru. Naučil som sa efektívnejšie komunikovať s ľuďmi pracujúcimi v tíme. Uvedomil som si, že programovanie nie je iba o písaní kódu, ale aj o plánovaní a algoritmickom myslení. Ďalej som zistil, že písať kód "čisto" nie je také jednoduché, ako sa môže na prvý pohľad zdať. Zistil som, že je tiež veľmi dôležité dokázať porozumieť zadaniu a pochopiť biznis modelu klienta tak, aby boli jeho požiadavky na produkt správne splnené.

Moju odbornú prax považujem za skvelé zakončenie môjho bakalárskeho štúdia a hodnotím ju veľmi kladne. Dostal som možnosť vyvíjať produkt pre reálnych klientov a pre reálne využitie. Viem, že tento produkt sa bude používať ešte dlho a som rád, že som sa na ňom mohol podieľať. Na záver by som chcel podotknúť, že táto odborná prax bola veľkým prínosom pre môj osobný kariérny rast.

8 Použitá literatura

[1] Laravel, framework jazyka php [online]. [cit. 2020-05-05]. Dostupné z: https://www.itnetwork.cz

[2] Laravel, framework jazyka php [online]. [cit. 2018-05-05]. Dostupné z: https://laravel.com

[3] Vue.js, framework jazyka javascript [online]. [cit. 2018-05-05]. Dostupné z: https://starkmedia.cz

[4] Vue.js, framework jazyka javascript [online]. [cit. 2018-05-05]. Dostupné z: https://www.itlearning.sk

[5] Docker virtualizácia [online]. [cit. 2018-05-05]. Dostupné z: https://www.abclinuxu.cz

[6] Docker virtualizácia [online]. [cit. 2018-05-05]. Dostupné z: https://notebooks-forge.readthedocs.io

[7] Docker virtualizácia [online]. [cit. 2018-05-05]. Dostupné z: https://santhoshravirala.com/docker/

[8] MySQL databázový systém [online]. [cit. 2018-05-05]. Dostupné z: https://best-hosting.cz